Conditional Cube Key Recovery Attack on Round-Reduced Xoodyak

Mohammad Vaziri(✉) and Vesselin Velichkov

The University of Edinburgh, Edinburgh, UK
{mohammad.vaziri,vvelichk}@ed.ac.uk

Abstract. Since the announcement of the NIST call for a new lightweight cryptographic standard, a lot of schemes have been proposed in response. Xoodyak is one of these schemes and is among the finalists of the NIST competition with a sponge structure very similar to the Keccak hash function – the winner of the SHA3 NIST competition. In this paper with conditional cube attack technique, we fully recover the key of Xoodyak reduced to 6 and 7 rounds with time complexity resp. $2^{42.58}$ and $2^{76.003}$ in the nonce-reusing scenario. In our attack setting, we import the cube variables in the absorbing associated data phase, which has higher degree of freedom in comparison to data absorption phase. We use MILP tool for finding enough cube variables to perform the conditional key recovery attack. The 6-round attack is practical and has been implemented. To the best of our knowledge, this is the first proposed attack on 7-round Xoodyak.

Keywords: Xoodyak · Symmetric-key · Cryptanalysis · Conditional Cube Attack · Lightweight Cryptography · MILP

1 Introduction

Due to the importance of lightweight cryptographic algorithms, the US National Institute of Standards and Technology (NIST) in 2018 [18] announced an open call for new lightweight cryptographic standards. On March 29, 2021, NIST announced ten finalists and Xoodyak is one of the finalists. On February 7, 2023, NIST announced Ascon [11] as the winner of the competition [17]. Even though Xoodyak was not chosen as the winner it still remains of interest as a secure and efficient lightweight algorithm against which no attack on the full version has been reported to date. In this paper we improve the best known attack on Xoodyak reduced to 6 rounds and propose the first 7-round attack.

Xoodyak [6] is a scheme based on the Xoodoo permutation, proposed by the same research team [5] that designed SHA3/Keccak [10] and has a very similar structure. Throughout this paper, for the sake of convenience, the expression

Part of this work was done during a research visit of the first author to the Digital Security (DiS) Group at Radboud University Nijmegen.

J. Zhou et al. (Eds.): ACNS 2023 Workshops, LNCS 13907, pp. 43–62, 2023.
https://doi.org/10.1007/978-3-031-41181-6_3

SHA3-like designs include all of the variants that are derived from the Keccak sponge-based round function or from the Xoodoo permutation.

Cube attacks are a class of symmetric-key attacks, targeting cryptographic primitives with low algebraic degree. In the attack procedure, the cryptographic primitive is interpreted as a polynomial with algebraic degree n. The attack leverages the fact that summing the output of 2^{n-1} inputs composed of all possible values of a set (i.e., a cube) of $n-1$ variables (all other variables being fixed), yields a linear function. If the obtained linear function contains some secret information (e.g. bits from the internal state or the key), the secret information can be recovered by solving a linear system of equations.

To improve the efficiency of the original cube attack, two types of related attacks have been proposed. The first one is called a cube attack-like attack and for the first time was introduced by Dinur et al. [7] at EUROCRYPT 2015. Later in [4,12,19] the authors applied cube attack-like technique to attack different variants of SHA3-like designs. The second semi-cube attack is called the conditional cube attack and for the first time was introduced by Huang et al. ([13]) at EUROCRYPT 2017 and further extended by [15,16,20,22].

Cube attack-like cryptanalysis is similar to the original cube attack and has an offline and an online phase. In the offline phase, a dictionary of all values of some key bits (so-called related keys) and the corresponding value of the output cube sum after certain number of rounds is compiled. In the online phase, by having the real value of the related key bits, the cube sum is computed and one looks for a match in the dictionary. In the conditional cube attack, the value of the cube sum should become zero under certain conditions and by guessing the key bits involved in the conditions, those key bits are recovered.

The proposed attacks in [7], are one of the first works on analyzing the resistance of SHA3-like designs against the cube attack. The authors implemented various types of algebraic attacks on keyed Keccak and Keyak [1][1] variants and broke 6–7 rounds of these variants in practical time. Inspired by [3], with setting up cube variables in the column parity, they bypassed the propagation produced by θ in the first round.

In [13], Huang et al. for the first time, developed a new type of conditional cube tester for SHA3-like designs. Inspired by dynamic cube attack [9], the authors imposed some bit conditions for certain cube variables for controlling the propagation of cube variables caused by the nonlinear operation χ and constructed a cube tester with smaller dimensions. They applied their model to reduced-round Keccak-MAC [3] and Keyak [1] and improved the previous proposed results for recovering the key, in terms of complexity and number of rounds.

Later in [15], by using a MILP tool, the authors improved the key recovery attack in [13] on reduced-round Keccak-MAC-384 and Keccak-MAC-512 by 1 round. For increasing the number of rounds of the attack, the challenge was finding enough ordinary cube variables. For being able to find enough ordinary cube variables, the MILP method proposed by the authors described the relations between ordinary cube variables and conditional cube variables in the first and

[1] Keyak is an AE scheme based on Keccak sponge function.

second rounds. In [20], for improving the conditional cube attacks on SHA3-like designs in terms of the complexity and number of rounds, the authors proposed another MILP model, which accurately described the propagations of the cube variables through the two rounds and also proposed a MILP model which was able to linearize first two rounds of SHA3-like designs. The authors applied their technique to attack against KMAC128 and KMAC256 [21].

Some variants of SHA3-like designs such as Ketje Jr [2], Xoodyak, do not have enough degree of freedom to perform the Huang et al.'s technique [13], even with the help of MILP tool. To be more specific, the portion of their state allocated for importing the data is not large enough to put enough cube variables to perform the attack. For dealing with a low degree of freedom in some variants of SHA3-like designs, Li et al. [16] introduced the so-called kernel quadratic term, which was more compatible with such variants.

There exist just 2 works in the area of cube key-recovery attack on Xoodoo structures. The first one is cube attack-like cryptanalysis on 6 rounds of a Xoodoo-based authenticated encryption proposed by [19] with time complexity and memory complexity resp. 2^{89} and 2^{55}. The second one is a conditional cube attack on 6 rounds of Xoodyak proposed by [22] with $2^{43.8}$ time complexity and negligible memory cost.

In this paper by applying Li et al.'s conditional cube attack technique [16], we recover the key of 6 and 7-round reduced Xoodyak. Unlike others [16,22], we import the cube variables in the absorbing associated data phase instead of the absorbing data phase. [2] Table 1 shows related results in comparison to our results.

Table 1. Summary of key recovery attacks on SHA3-like designs

Variant	Capacity	Degree of Freedom	Rounds	Time	Reference
Xoodyak	192	192	6	$2^{43.8}$	[22]
Xoodyak	192	352	6	$2^{42.45}$	Sect. 2
Xoodyak	192	352	7	$2^{76.003}$	Sect. 3

Our Contributions. In this paper, we apply the Li et al.'s conditional cube attack technique [16] on Xoodyak and present the first 7 rounds of attack on Xoodyak, and also 6 rounds with lower complexity in comparison to the proposed attack in [22]. The complexities of our attack are listed in Table 1, and the details are as follows:

- For 6-round Xoodyak in nonce-misuse settings, we recover the 384-bit key with time complexity $2^{42.45}$ and negligible memory cost.
- For 7-round Xoodyak in nonce-misuse settings, we recover the 384-bit key with time complexity $2^{76.003}$ and negligible memory cost.

[2] the corresponding source code is available via https://github.com/mohammadvaziri/Conditional-Cube-Attack-on-Xoodyak.

Organization. In Sect. 2 some notations and brief descriptions of Xoodyak are given. Some related works including cube attack and conditional cube attack techniques are introduced in Sect. 3. Section 4 describes the MILP search model for conditional cube attack. Section 5 gives the applications to Xoodyak. Section 6 concludes this paper.

2 Preliminaries

2.1 Notations

Throughout the paper, we will use the following notations for Xoodoo:

S_0	initial state of the Xoodoo permutation,
$S_{i-1,\theta}$	internal state of Xoodoo after θ in the i-th round,
$S_{i-1,\rho_{west}}$	internal state of Xoodoo after ρ_{west} in the i-th round,
$S_{i-1,\chi}$	internal state of Xoodoo after χ in the i-th round,
S_i	output state of Xoodoo of the i-th round,
$(i, *, k)$	index of a column,
$(*, j, k)$	index of a row,
$(i, j, *)$	index of a lane,
(i, j, k)	index of a bit,
$A[i][j]$	the lane indexed by $(i, j, *)$ of state A,
$A[i][j][k]$	the bit indexed by (i, j, k) of state A.

2.2 Description of Xoodyak

Xoodyak [6] is designed based on Xoodoo [5] permutation which includes an Authenticated Encryption with Additional Data (AEAD) scheme and a hashing scheme. Xoodoo has a 384-bit state A, which is represented as a three-dimensional array of bits, namely $A[4][3][32]$. In the state, the one-dimensional arrays $A[x][*][z]$, $A[*][y][z]$ and $A[x][y][*]$ are called a column, a row, and a lane respectively. The coordinates are computed modulo 4, modulo 3, and modulo 32 for x, y, and z respectively. The round function of Xoodoo is as $R = \rho_{east} \circ \chi \circ \iota \circ \rho_{west} \circ \theta$ and in the following the details of the round function of Xoodoo permutation based on the order are given:

$$\theta : A[x][y][z] = A[x][y][z] \bigoplus \sum_{j=0}^{2} (A[x-1][j][z-5] \bigoplus A[x-1][j][z-14]).$$
$$\rho_{west} : A[x][1][z] = A[x-1][1][z], A[x][2][z] = A[x][2][z-11].$$
$$\iota : A[0][0] = A[0][0] \oplus RC_i.$$
$$\chi : A[x][y][z] = A[x][y][z] \oplus ((A[x][y+1][z] \oplus 1) \wedge A[x][y+2][z]).$$
$$\rho_{east} : A[x][1][z] = A[x][1][z-1], A[x][2][z] = A[x-2][2][z-8].$$

In Fig. 1, f is the 12-round permutation Xoodoo. As shown in Fig. 1, the structure of the Xoodyak-AEAD is as follows: first, a 128-bit key is absorbed into the 384-bit state, then the Xoodoo permutation is applied to the state;

second, a 128-bit nonce is absorbed then the Xoodoo permutation is applied to the state; third, the 352-bit associated data block is absorbed, then the Xoodoo permutation is applied to the state; then it starts to absorb 192-bit plaintext blocks and output ciphertext blocks in each term; the finalization phase happens at the end. More information on the details of the Xoodyak-AEAD can be found in [6].

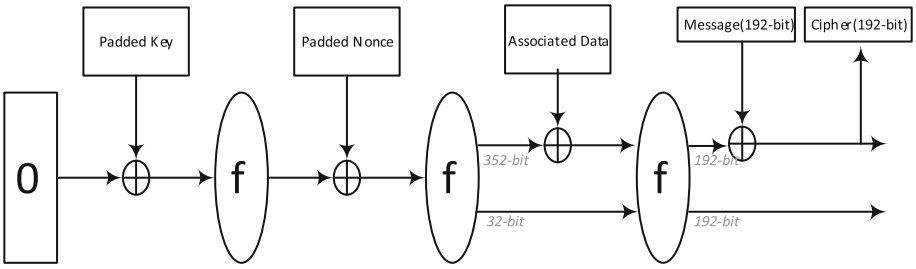

Fig. 1. Xoodyak-AEAD structure

Our attack has two essential characteristics: firstly, we assume that the nonce is reused, as the same assumption used by [16]; secondly, we apply our conditional cube attack in the phase of absorbing associated data, contrary to [16], where the attack is applied in the absorbing data phase. The degree of freedom in the absorbing data phase is 192-bit. Following [6], however, the degree of freedom is 352-bit. To be more specific, according to Sect. 2.3 in [6], the absorb rate R_{kin} is 44 bytes and according to Algorithms 1 and 2 in [6], R_{kin} is used during the absorbing of associated data, which means the degree of freedom in associated data phase is 352-bit. It should be noted that in order to recover the main key, in our attack scenario the full 384-bit state needs to be recovered and the key can be computed inversely, while in the scenario of [16] just half of the state needs to be recovered.

3 Review of Cube Attacks and Related Techniques

In this section, we briefly describe the cube attack and overview 2 types of proposed conditional cube attacks on the structure of the SHA3-like designs. The first type of conditional cube attack is proposed by Huang et al. [13] to attack Keccak keyed mode, and the second type is introduced by Li et al. [16] to attack the schemes that have a lower degree of freedom.

3.1 Cube Attack

Cube attack is a chosen plaintext attack, which can be seen as an extension of higher order differential attacks [14], and was formally introduced by Dinur

et al. [8]. In the cube attack, the output of the cipher is regarded as a black box polynomial and by querying the black box polynomial a linear system of equations is created. The linear system of equations contains some (or whole) part of the secret key and by solving this system some (or whole) part of the secret key is recovered. Cube attack is based on the following theorem:

Theorem 1 ***(Dinur, Shamir [8]).*** *Given a polynomial $f : X^n \rightarrow \{0,1\}$ of degree d. Suppose that $0 < k < d$ and t is the monomial $x_0...x_{k-1}$. Write the function as*

$$f(x) = t \cdot P_t(x) + Q_t(x)$$

where none of the terms in $Q_t(x)$ is divisible by t. Note that $degP_t \leq d-k$. Then the sum of f overall values of the cube (defined by t) is

$$\sum_{x'=(x_0,...,x_{k-1})\in C_t} f(x', x) = P_t(\underbrace{1, ..., 1}_{k}, x_k, ..., x_{n-1})$$

whose degree is at most $d-k$ (or 1 if $k = d-1$), where the cube C_t contains all binary vectors of the length k.

In Theorem 1, $P_t(x)$ is the "superpoly" of the term t. The original cube attack contains two phases resp. offline and online. In the offline phase, by selecting a random cube set, the superpoly, independent of the secret variables, is obtained and in the online phase, by fixing the secret variables the actual value of the obtained Superpolies is computed, and some linear system of equations is built. As can be seen in [7,8], implementing the original cube attack is very expensive, since the cube variables in the cube set are randomly selected and most of the time the obtained Superpolies do not contain any useful information.

3.2 Conditional Cube Attack

To perform the conditional cube attack on r round of the SHA3-like designs, the attacker looks for a set of cube variables $\{v_0, ..., v_{k-1}\}$, $k \geq 2$ such that the term $v_0...v_{k-1}$ does not appear at the output of the $r-th$ round if some conditions imposed at the first round hold. For doing so, some cube variables are determined as conditional cube variables and for controlling the propagation of the conditional cube variables, the conditions are imposed on the neighboring positions of the conditional cube variables prior to the first χ operation. Therefore, if the imposed conditions hold, the propagation of the conditional cube variables are controlled and the term $v_0...v_{k-1}$ does not appear at the output of the $r-th$ round, otherwise, it does appear. Consequently, the cube sum is zero only if the imposed conditions hold. Since the position of the conditions includes the summation of some secret key bits and constant bits, the conditions with high probability hold, if only the key bits involved in the conditions will be guessed correctly.

Therefore, for performing a conditional cube attack on r rounds of a SHA3-like design, the attacker first tries to find a term that does not appear at the

output of the r-th round under certain conditions, then guesses the involved secret key bits in the conditions. If the cube summation is zero, then the guessed key bits with high probability are correct. In the following 2 techniques of conditional cube attack proposed by Huang et al. and Li et al. is presented.

Huang et al.'s Conditional Cube Attack. The first type of conditional cube attack is introduced by Huang et al. [13]. For performing the conditional cube attack, first, they define two types of cube variables as *conditional cube variable* and *ordinary cube variable*. For attacking r rounds of a SHA3-like design, they prove that by having $p(0 \leq p < 2^{n+1})$ conditional cube variable and $q(q = 2^{n+1} - 2p + 1)$ ordinary cube variables, multiplication of conditional and ordinary cube variables will not appear at the output of the r-th round. In the following, the details of the Huang et al.'s conditional cube attack technique are given.

Definition 1 *([13]). Cube variables that have propagation controlled in the first round and are not multiplied with each other after the second round of Keccak are called* ***conditional cube variables****. Cube variables that are not multiplied with each other after the first round and are not multiplied with any conditional cube variable after the second round are called* ***ordinary cube variables****.*

Theorem 2 *([13]). For $(n+2)$-round Keccak sponge function $(n > 0)$, if there are p $(0 \leq p < 2^{n+1})$ conditional cube variables $v_0, ..., v_{p-1}$, and $q = 2^{n+1} - 2p + 1$ ordinary cube variables, $u_0, ..., u_{q-1}$ (If $q = 0$, we set $p = 2^{n+1}$), the term $v_0 v_1 ... v_{p-1} u_0 ... u_{q-1}$ will not appear in the output polynomials of $(n+2)$- round Keccak sponge function.*

Theorem 2 indicates that by having a certain amount of conditional cube variables and ordinary cube variables, the multiplication of those does not appear at the output of the certain round of SHA3-like designs.

It is worth mentioning as the number of conditional cube variables is increased the number of conditions is increased, as well, which leads to higher time complexity in recovering the key. Therefore, in most of the works they have used just one conditional cube variable [13,15]. In the case of using one conditional cube variable, the following corollary is used:

Corollary 1. *For $(n+2)$-round Keccak sponge function $(n > 0)$, if there is one conditional cube variable v_0, and $q = 2^{n+1} - 1$ ordinary cube variables, $u_0, ..., u_{q-1}$, the term $v_0 u_0 ... u_{q-1}$ will not appear in the output polynomials of $(n+2)$-round Keccak sponge function.*

The attack procedure of the Huang et al.'s conditional cube attack technique [13], which was later extended with an improved MILP model by Song et al. [20] can be summarized in the following steps:

1. Determine the position of the conditional cube variable(s) at initial state.
2. Use the MILP model proposed by Song et al. [20] to find position of the bit conditions and also the ordinary cube variables satisfying the requirements in Theorem 2.

3. Determine the key bits involved in the bit conditions.
4. Put the value 0 for any constant position at the initial state and for each possible value of the bit conditions compute the cube sum.
5. If the cube sum is zero, then the guessed values of the bit conditions are correct.

Li et al.'s Conditional Cube Attack. Huang et al.'s conditional cube attack [13] becomes invalid for some variants of SHA3-like designs with a lower degree of freedom to find enough ordinary cube variables. To overcome the challenge, Li et al. [16] came up with a new type of conditional cube attack. In their technique instead of using the conditional cube variable, they introduce a degree two term called *kernel quadratic term*. In the following, the definition of the kernel quadratic term and the related corollary for performing the conditional cube attack is presented.

Definition 2 *([16]). Suppose all the (q+2) cube variables are* $v_0, v_1, u_0, ..., u_{q-1}$*, and constraints are as follows:*

- *After the first round,* v_0v_1 *is the only quadratic term.*
- *In the second round, if the bit conditions are satisfied,* v_0v_1 *does not multiply with any of* $u_0, ..., u_{q-1}$*, i.e. no cubic term occurs.*
- *In the second round, if the bit conditions are not satisfied,* v_0v_1 *multiplies with some of* $u_0, ..., u_{q-1}$*, i.e. some cubic terms like* $v_0v_1u_i (i = 0, ..., q-1)$ *occur.*

Then v_0v_1 *is called* ***kernel quadratic term****. The remaining cube variables except* v_0 *and* v_1*, i.e.* $u_0, ..., u_{q-1}$*, are called* ***ordinary cube variables****.*

From the Definition 2, the following corollary can be deduced.

Corollary 2 *([16]). For (n+2)-round Keccak sponge function (n > 0), if there is one kernel quadratic term* v_0v_1*, and* $q = 2^{n+1} - 1$ *ordinary cube variables,* $u_0, u_1, ..., u_{q-1}$*, the term* $v_0v_1u_0u_1, ..., u_{q-1}$ *will not appear in the output polynomials of (n + 2)-round Keccak sponge function under certain bit conditions. So the distinguisher is approached as follows:*

- *under right bit conditions, the degree of output polynomials of* $n + 2$ *rounds is no more than* 2^{n+1}*.*
- *under the wrong bit conditions, the degree of output polynomials of* $n + 2$ *rounds is* $q + 2 = 2^{n+1} + 1$*.*

The attack procedure of the Li et al.'s conditional cube attack technique can be summarized in the following steps:

1. Determine the position of the kernel quadratic term at initial state.
2. Use the MILP model proposed by Song et al. [20], customized by Li et al. [16] to find position of the bit conditions and also the ordinary cube variables satisfying the requirements in Corollary 2.
3. Determine the key bits involved in the bit conditions.
4. Put the value 0 for any constant position at the initial state and for each possible value of the bit conditions compute the cube sum.
5. If the cube sum is zero, then the guessed values of the bit conditions are correct.

Comparison of Huang et al.'s and Li et al.'s Technique. In Li et al.'s technique, the kernel quadratic term is set in column parity at the output of the first round. By doing so, the propagation produced by θ at the second round is bypassed on the kernel quadratic term. Therefore, for satisfying the conditions mentioned in Corollary 2, all we need is to avoid having ordinary cube variables in 4 neighboring positions of the kernel quadratic term before the second χ. In Huang et al.'s technique, however, the second θ propagates the conditional cube variable, so the positions that the ordinary cube variables should not appear before the second χ is more than four[3]. This strategy in Li et al.'s technique has made the challenge of finding enough ordinary cube variables easier.

Comparison of Conditional Cube Attack and Original Cube Attack. In the original cube attack, the cube variables are randomly chosen and there is no guarantee on having a superpoly containing some secret key bits. In the conditional cube attack, however, the cube variables are carefully selected and the imposed conditions, always contain secret key bits. Moreover, the original cube attack contains two offline and online phases and consumes a huge memory, while the conditional cube attack is just 1 online phase and memory consumption is negligible.

4 Conditional Cube Attacks on Round-Reduced Xoodyak

As we mentioned, in our attack scenario we insert the cube variables in the associated data phase, and as Fig. 2 shows, the `XOR` of the associated-data state and unknown-data state is passed through the Xoodoo permutation. Therefore, for recovering the master key, the whole state of the unknown data should be recovered.

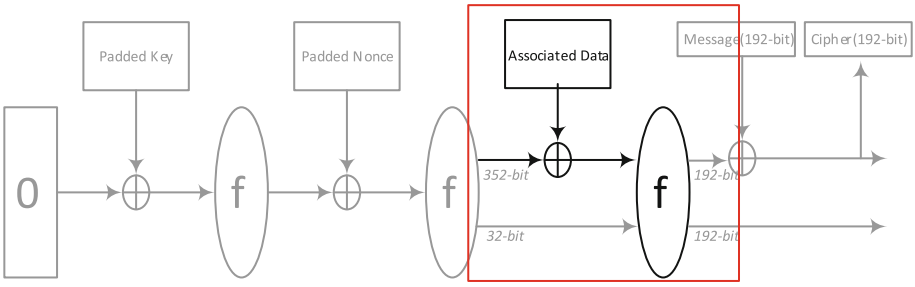

Fig. 2. Cube variables are injected in the associated data state. Our attack is applied at the stage within the red frame. (Color figure online)

For finding a distinguisher, we use Li et al.'s technique. We also applied Huang et al.'s technique but the MILP tool returned "infeasible solution". We

[3] By setting up the conditional cube variable in the column parity in the first round, In Keccak sponge function is 22 and in Xoodoo permutation is 14.

suspect that the reason is insufficient degrees of freedom, but further analysis is necessary to confirm this. In the following, the way of finding the positions of the cube variable in the initial state for building the Kernel quadratic term, and the way of constructing the MILP model for having enough ordinary cube variables are explained.

4.1 Finding Kernel Quadratic Term

The best way to control the propagation of the quadratic term is by setting the kernel quadratic term in the column parity at the beginning of the second round. To do this, in [16], they propose a reversing technique in which first they assume the positions of the kernel quadratic term are set in the column parity i.e. two kernel quadratic term (v_0v_1) exist at the same column at the output of the first round, then they reversely impose the operations of the SHA3-like design.

In order to represent the number of the active bits of a conditional cube variable (kernel quadratic term) in [13] a pattern as x-y-z is introduced, where x, y and z are the number of active bits of a conditional cube variable (kernel quadratic term) in the initial state, the output of first round and the input of second χ operation, respectively. In [16], they propose a 6-2-2 pattern for the propagation of the kernel quadratic term v_0v_1, which acts different than the pattern 2-2-22 proposed by [13], for the propagation of the conditional cube variable v_0. In the 2-2-22 pattern the propagation of the conditional cube variable v_0 is controlled in the first round, but the θ operation in the second round diffuse v_0 in 22 positions, which means the ordinary cube variables should not appear in 44 positions at the input of the second χ operation[4]. However, in the 6-2-2 pattern, the ordinary cube variables should not appear in just 4 positions at the input of the second χ operation.

In [22], instead of using 6-2-2 pattern, they offer to use 8-2-2 pattern. Since in 6-2-2 pattern, for creating the v_0v_1, the cube variable v_0 is not set in column parity, and as the ordinary cube variables are not supposed to multiply with v_0, the degree of freedom for the ordinary cube variables is decreased. However, in 8-2-2 pattern, both of the cube variables v_0 and v_1 are set in column parity, which creates more degree of freedom for the selection of the ordinary cube variables.

In our case, for finding the positions of v_0 and v_1 in the kernel quadratic term v_0v_1, we also use 8-2-2 pattern, and at the same time by following the reversing technique proposed by [16], we try to avoid having the variables v_0 and v_1 in the last lane. The obtained positions for the v_0 and v_1 at the initial state S_0 are as $A[0][1][0] = A[0][2][0] = A[1][0][20] = A[1][2][20] = v_0$ and $A[1][0][21] = A[1][2][21] = A[1][0][31] = A[1][1][31] = v_1$. According to Fig. 3, the kernel quadratic term v_0v_1 appears in the positions $S_1[1][0][0]$ and $S_1[1][1][0]$.

[4] To avoid having multiplication, the ordinary cube variables should not appear in the neighboring positions of the conditional cube variable.

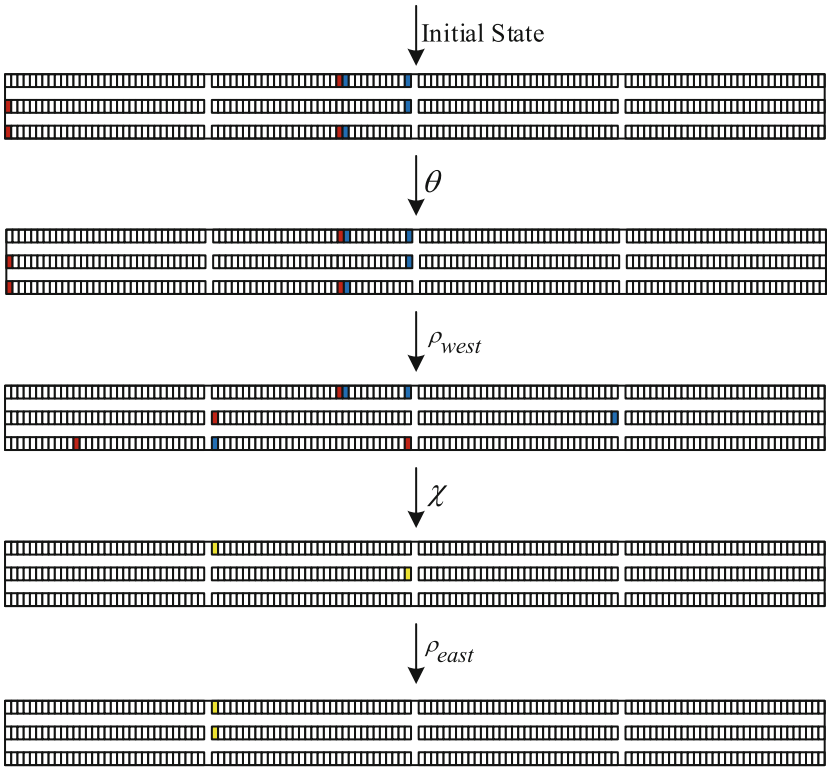

Fig. 3. Forming of the kernel quadratic term in the first round of Xoodyak. Red, blue and yellow squares show the positions of v_0, v_1, and v_0v_1 respectively. (Color figure online)

4.2 MILP Model for Li et al. Technique

In this section, we customize a MILP model for our scenario proposed by Song et al. [20] and also used by [16,22]. This MILP model is meant to find $2^{n+1}-1$ ordinary cube variables to attack $n+2$ rounds of Xoodyak, by aiming to minimize the number of bit-conditions. We assume the bit positions of v_0 and v_1 at the kernel quadratic term v_0v_1 in S_0 are the same as we obtained in the previous section. The applied symbols in the model are as follows: $a[x][y][z] = 1, a'[x][y][z] = 1, b[x][y][z] = 1$ means state bit $S_0[x][y][z]$, $S_{0,\rho_{weast}}[x][y][z]$, $S_1[x][y][z]$ contains at least one ordinary cube variable, respectively. The MILP model determines the valid positions for the ordinary cube variables at S_0, $S_{0,\rho_{weast}}$ and $S_{1,\rho_{weast}}$, describes the treatment of θ and χ at the first round, and also determines the total number of the ordinary cube variables, needed to perform the attack.

Modeling the First Round. Due to the absorbing rate of the associated data state, the first set of constraints needs to be added to the MILP model to avoid having

any ordinary cube variable in the last lane of the initial state. The constraints 1 are meant for this purpose:

$$a[3][2][z] = 0, \quad 0 \leq z < 32 \tag{1}$$

The bit positions that are already taken by the cube variable v_0 and v_1, at S_0 should not be a potential position for the ordinary cube variables to be taken. To satisfy such constraints, the following equalities are added to the model.

$$\begin{aligned} &a[0][1][0] = a[0][2][0] = a[1][0][20] = a[1][2][20] = \\ &a[1][0][21] = a[1][2][21] = a[1][0][31] = a[1][1][31] = 0 \end{aligned} \tag{2}$$

For describing the treatment of the columns $(x, *, z)$ in θ operation in [20], two types of Boolean variables called $f[x][z]$ and $g[x][z]$ are introduced. If at the column $(x, *, z)$ at the initial state A, $g[x][z] = 1$, then the summation of all 3 bits of the column should be nonzero, and if $f[x][z] = 1$, despite the existence of some variable in the column $(x, *, z)$, the summation of all 3 bits of the column should be zero. It should be noted that $f[x][z] = 1$ means a 1-bit degree of freedom is consumed. According to [20], if setting all of the cube variables in column parity is of interest, $g[x][z]$ should be set to 0 for all of the columns. In our case, since we have enough degree of freedom and also for simplifying the MILP model, we set all of the cube variables in column parity. Hence, following [20], the final equations for modeling the θ operation are as follows:

$$\begin{aligned} &-f[x][z] \geq -1 \\ &-a[x][0][z] + f[x][z] \geq 0 \\ &-a[x][1][z] + f[x][z] \geq 0 \\ &-a[x][2][z] + f[x][z] \geq 0 \\ &a[x][0][z] + a[x][1][z] + a[x][2][z] - 2f[x][z] \geq 0 \end{aligned} \tag{3}$$

As in [16,22] is mentioned, the cube variables v_0 and v_1 should not multiply with any other ordinary cube variable. Since they are supposed to multiply with one another and build the kernel quadratic term. Therefore, for preventing the multiplication of any ordinary cube variables with v_0 and v_1, the neighboring positions of v_0 and v_1 at the state $S_{0,\rho_{weast}}[x][y][z]$ should be 0. According to Sect. 4.1, the cube variables v_0 and v_1 are in 8-bit positions. In total there exists 14 bit-positions in $S_{0,\rho_{weast}}[x][y][z]$ that should be equal to 0:

$$\begin{aligned} &a[0][0][11] = a[0][1][11] = a[1][1][20] = a[1][2][20] = \\ &a[1][1][21] = a[1][2][21] = a[2][0][31] = a[2][2][31] = \\ &a[1][0][0] = a[1][1][0] = a[1][2][0] = a[1][0][31] = \\ &a[1][1][31] = a[1][2][31] = 0 \end{aligned} \tag{4}$$

In [20], for modeling the χ operation, two other Boolean variables so called $v[x][y][z]$ and $h[x][y][z]$ are introduced for describing the bit conditions in which $v[x][y][z] = 1$ indicates the existence of a condition at the state bit $S_{0,\rho_{weast}}[x][y][z]$ that has to be fixed to the amount of $h[x][y][z]$. Following [20], if $a'[x][y][z]$ and $b[x][y][z]$ would represent the existence of the cube variables in

the input and output of the χ operation respectively, the constraints 5 describe the treatment of the χ operation:

$$\begin{aligned}
&-a'[x][y][z] - a'[x][y+1][z] \geq -1\\
&-a'[x][y][z] + b[x][y][z] \geq 0\\
&-a'[x][y+1][z] - v[x][y+1][z] \geq -1\\
&-a'[x][y+2][z] - v[x][y+2][z] \geq -1\\
&-a'[x][y][z] - a'[x][y+1][z] - h[x][y+2][z] + b[x][y][z] \geq -1\\
&-a'[x][y][z] - v[x][y+1][z] - h[x][y+1][z] - b[x][y][z] \geq -2\\
&a'[x][y][z] - v[x][y+2][z] + h[x][y+2][z] - b[x][y][z] \geq -1\\
&a'[x][y][z] + a'[x][y+1][z] + a'[x][y+2][z] - b[x][y][z] \geq 0\\
&-a'[x][y+1][z] - a'[x][y+2][z] + v[x][y+1][z] + v[x][y+2][z] + b[x][y][z] \geq 0\\
&-a'[x][y+1][z] - a'[x][y+2][z] + v[x][y+2][z] + h[x][y+1][z] + b[x][y][z] \geq 0
\end{aligned} \tag{5}$$

Modeling the Second Round. Preventing the multiplication of any ordinary cube variable with the kernel quadratic term is the only set of constraints that needs to be considered in the second round. Since the kernel quadratic term is set in the column parity, it appears just in two positions before the second χ i.e. $S_{1,\rho_{weast}}$. Therefore, we should avoid having any ordinary cube variable just in 4 neighboring positions of the kernel quadratic term in the same column.

Following the constraints proposed by [20] and used by [16,22], we need to avoid having uncertain propagation in each neighboring bit of the kernel quadratic term. In order to do so, for each neighboring bit position of the kernel quadratic terms, a new dummy variable $e_i, i = 0, 1, 2, 3$ is introduced. $e_i = 1$, if at least one ordinary cube variable exists in the related neighboring position.

To avoid having uncertain propagation at the neighboring positions of the kernel quadratic term, according to the θ operation, each neighboring bit at $S_{1,\rho_{weast}}$ contains a linear expression of 7 bits from S_1, and according to the χ operation in the first round, the existence of ordinary cube variable in each bit at S_1 is dependent on the existence of any ordinary cube variable in two neighboring bits, and corresponded bit at $S_{0,\rho_{weast}}$. As a result, for having certain propagation in the neighboring positions of the kernel quadratic term, for each variable $b[x][y][z]$ that exists in i-th neighboring positions of the kernel quadratic term, the inequalities 6 proposed by [20] is used.

$$\begin{aligned}
&-e_i - a'[x][y+1][z] - a'[x][y+2][z] \geq -2\\
&-e_i - a'[x][y+1][z] + v[x][y+2][z] \geq -1\\
&-e_i - a'[x][y+2][z] + v[x][y+1][z] \geq -1\\
&-e_i - a'[x][y+1][z] - v[x][y+1][z] \geq -2\\
&-e_i - a'[x][y+2][z] - v[x][y+2][z] \geq -2\\
&-e_i - a'[x][y][z] - a'[x][y+1][z] \geq -2
\end{aligned} \tag{6}$$

The constraints 6 indicate if in the i-th neighboring position of the quadratic term e_i is nonzero, the position contains a linear expression of some certain ordinary cube variable. For canceling the linear expression, a 1-bit degree of freedom is consumed at the initial state. Hence for computing the total number of ordinary cube variables, subtraction of the $\sum_{i=0}^{3} e_i$ is needed to be considered.

Modeling for Finding Ordinary Cube Variables. The steps to build a MILP model to find ordinary cube variables satisfying the conditions in the Definition 2, are as follows:

- The objective is minimizing the number of bit conditions. Since the variables $v[x][y][z]$ indicate the existence of a bit condition in $S_{0,\rho_{weast}}[x][y][z]$ the objective function is as follows:

$$\text{Minimize}: \sum_{i=0,j=0,k=0}^{i=3,j=2,k=31} v[x_i][y_j][z_k] \tag{7}$$

- Adding the constraints generated in the Sect. 4.2 for the first round.
- Adding the constraints generated in the Sect. 4.2 for the second round.
- For having $2^{n+1}-1$ ordinary cube variables to attack $n+2$ rounds of Xoodyak, the following constraint should be added to the model:

$$\sum_{i=0,j=0,k=0}^{i=3,j=2,k=31} a[x_i][y_j][z_k] - \sum_{i=0,k=0}^{i=3,k=31} f[x_i][z_k] - \sum_{i=0}^{i=3} e[x_i] = 2^{n+1} - 1 \tag{8}$$

  In the above equation, $\sum_{i=0,k=0}^{i=3,k=31} F[x_i][z_k] + \sum_{i=0}^{i=3} e[x_i]$ is the consumed degree of freedom in the initial state S_0.
- In order to avoid not having any bit condition, the following constraint will be added to the model:

$$\sum_{i=0,j=0,k=0}^{i=3,j=2,k=31} v[x_i][y_j][z_k] \geq 1 \tag{9}$$

5 Key Recovery on 6 and 7 Rounds Xoodyak

In this section, we apply the MILP model proposed in Sect. 4.2 for 6 and 7 rounds of Xoodyak respectively. Since the 6-round attack is practical, the full key recovery attack is implemented.

As we mentioned earlier in Sect. 4.1, the bit positions of the kernel quadratic term at the initial state are $A[0][1][0] = A[0][2][0] = A[1][0][20] = A[1][2][20] = v_0$ and $A[1][0][21] = A[1][2][21] = A[1][0][31] = A[1][1][31] = v_1$.

For recovering the key bits involved in the bit-conditions, firstly the key bits are guessed and if the cube sum would be zero, the guessed key bits are expected to be correct with high probability.

5.1 6 Rounds Conditional Cube Attack on Xoodyak

According to Fig. 2, we import the cube variables on the associated-data phase and recover the whole of the unknown state that is `XOR`ed with the associated data. The round function of the Xoodoo is reduced to 6 rounds.

Table 2. Parameters set for attack on 6-round Xoodyak

Kernel Quadratic term
$A[0][1][0] = A[0][2][0] = A[1][0][20] = A[1][2][20] = v_0$, $A[1][0][31] = A[1][1][31] = A[1][0][21] = A[1][2][21] = v_1$
Bit Conditions
$A[0][2][21] = k_{67} + k_{90} + k_{104} + k_{195} + k_{218} + k_{323} + k_{346} + n_{67} + n_{90} + n_{104} + n_{195} + n_{218} + n_{323} + 1$
Ordinary Cube Variables
$A[0][1][3] = u_0, A[0][2][3] = u_0, A[0][0][9] = u_1, A[0][1][9] = u_2,$ $A[0][2][9] = u_1 + u_2, A[0][0][12] = u_3, A[0][1][12] = u_3, A[0][0][23] = u_4,$ $A[0][1][23] = u_5, A[0][2][23] = u_4 + u_5, A[0][0][24] = u_6, A[0][1][24] = u_7,$ $A[0][2][24] = u_6 + u_7, A[0][0][30] = u_8, A[0][1][30] = u_8, A[1][0][2] = u_9,$ $A[1][2][2] = u_9, A[1][0][11] = u_{10}, A[1][1][11] = u_{10}, A[1][0][14] = u_{11},$ $A[1][2][14] = u_{11}, A[1][0][22] = u_{12}, A[1][2][22] = u_{12}, A[2][0][2] = u_{13},$ $A[2][1][2] = u_{14}, A[2][2][2] = u_{13} + u_{14}, A[2][0][5] = u_{15}, A[2][1][5] = u_{16},$ $A[2][2][5] = u_{15} + u_{16}, A[2][1][8] = u_{17}, A[2][2][8] = u_{17}, A[2][0][9] = u_{18},$ $A[2][1][9] = u_{19}, A[2][2][9] = u_{18} + u_{19}, A[2][1][11] = u_{20}, A[2][2][11] = u_{20},$ $A[2][0][12] = u_{21}, A[2][1][12] = u_{22}, A[2][2][12] = u_{21} + u_{22}, A[2][0][14] = u_{23},$ $A[2][1][14] = u_{24}, A[2][2][14] = u_{23} + u_{24}, A[2][1][16] = u_{25}, A[2][2][16] = u_{25},$ $A[2][0][24] = u_{26}, A[2][1][24] = u_{27}, A[2][2][24] = u_{26} + u_{27}, A[2][0][30] = u_{28},$ $A[2][1][30] = u_{28}, A[3][0][1] = u_{29}, A[3][1][1] = u_{29}, A[3][0][25] = u_{30},$ $A[3][1][25] = u_{30}$
Guessed Key Bits
$k_{67} + k_{90} + k_{104} + k_{195} + k_{218} + k_{323} + k_{346}$

By linearizing 1 round of Xoodoo, the degree of the 6th round is 32. Therefore, in total, we need 33 cube variables to mount the attack, 2 of them build the kernel quadratic term and the rest are ordinary cube variables. Table 2 presents kernel quadratic term, ordinary cube variables, and bit conditions obtained by the MILP model.

According to Table 2, the only key bit needs to be recovered is as $k_{12} + k_{21} + k_{58} + k_{140} + k_{149} + k_{268} + k_{277}$. The time complexity to recover a 1-bit key is 2^{34}. Since the Xoodoo permutation is symmetric in z-axis, the key-bit $k_{12+i} + k_{21+i} + k_{58+i} + k_{140+i} + k_{149+i} + k_{268+i} + k_{277+i}$ can be recovered by i-bit $0 \leq i < 32$ rotating all of the parameters in z-axis. We can recover 352 bits by finding a different set of cube variables and also rotating them in z-axis. If we leave 32 remaining key-bit as an exhaustive search, the complexity for recovering 352 key-bit is $2^{8.45}2^{34} = 2^{42.45}$. As a result, the total complexity is $2^{42.45}+2^{32} \simeq 2^{42.451}$. This complexity is a bit lower than the proposed complexity by [22] i.e. $2^{43.8}$.

In our experiments, we generate 384 random bits as unknown data, which are to be XOR-ed with the associated data state, and in the associated data state we import the cube variables and set the rest of the positions to be zero. We calculate the cube sum on all of the lanes. It is worth mentioning that the probability of

having zero cube sum for all of the lanes for a random function is 2^{-384}. Hence, it is safe to declare the guessed key as correct with high probability, once the 33-dimension cube sums become zero.

To test the correctness of the parameters in Table 2, we run experiments, whose steps are summarized in Algorithm 1.

Algorithm 1. Testing the obtained parameters in Table 2

input: parameters in Table 2.
output: resulting cube sum for each guess.
Generate 384 random bits as the representation of the unknown data and store the state as Key;
Compute the value of $k_{67} + k_{90} + k_{104} + k_{195} + k_{218} + k_{323} + k_{346}$ and store it as the right key which needs to be recovered;
for each key guess $i, i \in \{0x0, 0x1\}$ **do**
  Initial-State = Key
  Final-State = 0
  Initial-State[10][26] = $i \oplus 1$
  for each possible value of cube set $j, j \in [0, 2^{34}]$ **do**
    Temp-State = Initial-State
    generate the j-th permutation of cube variables, and store it in Cubes-State
    Temp-State = Temp-state $\oplus$ Cubes-State
    Impose 6-round Xoodoo permutation on Temp-state
    Final-State = Final-state $\oplus$ Temp-State
  end for
  Print Final-State
end for

Next we present the results from experiments for testing the correctness of the parameters given in Table 2.

384-bit key K:

011001101100110100001010101111000 110100011010010000101101010010001
100100011101101100110100010010011 110101110101110110011000100111100
001010101110001001110111010100011 101011000011001000100110001100001
010100010100110110111111110010011 010010101101010010101110101000100
111100110100010110101100110101010 110111011101001100111100000000001
100000011001010101010000110110101 001001101110101111000011000000010

According to Table 2 and above generated key, the correct value for the guessed key bit is 1.

guessed value: 0, cube sums: `0x10b7a9c8, 0x189ab404, 0x3eeeacfa, 0x3eeeacfa, 0x5c5f184e, 0xd4497ce3, 0xff28ba81, 0xfa3d438c, 0x5155d784, 0x4aca9eb9, 0x748c5ed8, 0x53899c47, 0xad5b47e9`

guessed value: 1, cube sums: `0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0`

The program was run in Ubuntu 20.04 operating system with gcc 9.40. Recovery of a 1-bit key needs about 10 h using one CPU core (AMD EPYC 3.3GHz).

Table 3. Parameters set for attack on 7-round Xoodyak

Kernel Quadratic term
$A[0][1][0] = A[0][2][0] = A[1][0][20] = A[1][2][20] = v_0,$ $A[1][0][31] = A[1][1][31] = A[1][0][21] = A[1][2][21] = v_1$
Bit Conditions
$A[0][2][13] = k_4 + k_{13} + k_{50} + k_{132} + k_{141} + k_{260} + k_{269} + n_4 + n_{13} + n_{50} + n_{132} + n_{141} + n_{260}$
$A[0][2][22] = k_{13} + k_{22} + k_{59} + k_{141} + k_{150} + k_{269} + k_{278} + n_{13} + n_{22} + n_{59} + n_{141} + n_{150} + n_{269}$
$A[0][2][27] = k_{18} + k_{27} + k_{146} + k_{155} + k_{160} + k_{274} + k_{283} + n_{18} + n_{27} + n_{146} + n_{155} + n_{160} + n_{274} + 1$
$A[1][2][7] = k_2 + k_{25} + k_{130} + k_{153} + k_{258} + k_{281} + k_{295} + n_2 + n_{25} + n_{130} + n_{153} + n_{258} + n_{281}$
$A[1][2][16] = k_2 + k_{11} + k_{130} + k_{139} + k_{258} + k_{267} + k_{304} + n_2 + n_{11} + n_{130} + n_{139} + n_{258} + n_{267}$
Ordinary Cube Variables
$A[0][0][1] = u_0, A[0][1][1] = u_1, A[0][2][1] = u_0 + u_1, A[0][0][2] = u_2,$ $A[0][1][2] = u_2, A[0][1][9] = u_3, A[0][2][9] = u_3, A[0][0][10] = u_4,$ $A[0][1][10] = u_5, A[0][2][10] = u_4 + u_5, A[0][1][13] = u_6, A[0][2][13] = u_6,$ $A[0][0][14] = u_7, A[0][1][14] = u_8, A[0][2][14] = u_7 + u_8, A[0][0][15] = u_9,$ $A[0][1][15] = u_{10}, A[0][2][15] = u_9 + u_{10}, A[0][0][18] = u_{11}, A[0][1][18] = u_{12},$ $A[0][2][18] = u_{11} + u_{12}, A[0][1][19] = u_{13}, A[0][2][19] = u_{13}, A[0][0][27] = u_{14},$ $A[0][1][27] = u_{15}, A[0][2][27] = u_{14} + u_{15}, A[0][0][28] = u_{16}, A[0][1][28] = u_{16},$ $A[1][0][6] = u_{17}, A[1][2][6] = u_{17}, A[1][0][11] = u_{18}, A[1][2][11] = u_{18},$ $A[1][0][12] = u_{19}, A[1][2][12] = u_{19}, A[1][1][19] = u_{20}, A[1][2][19] = u_{20},$ $A[1][0][24] = u_{21}, A[1][2][24] = u_{21}, A[1][0][25] = u_{22}, A[1][2][25] = u_{22},$ $A[1][0][26] = u_{23}, A[1][2][26] = u_{23}, A[1][0][29] = u_{24}, A[1][2][29] = u_{24},$ $A[2][0][1] = u_{25}, A[2][1][1] = u_{26}, A[2][2][1] = u_{25} + u_{26}, A[2][0][2] = u_{27},$ $A[2][1][2] = u_{28}, A[2][2][2] = u_{27} + u_{28}, A[2][0][6] = u_{29}, A[2][1][6] = u_{30},$ $A[2][2][6] = u_{29} + u_{30}, A[2][1][7] = u_{31}, A[2][2][7] = u_{31}, A[2][0][10] = u_{32},$ $A[2][1][10] = u_{23}, A[2][2][10] = u_{23} + u_{32}, A[2][0][11] = u_{33}, A[2][1][11] = u_{34},$ $A[2][2][11] = u_{33} + u_{34}, A[2][1][12] = u_{35}, A[2][2][12] = u_{35}, A[2][0][14] = u_{36},$ $A[2][1][14] = u_{36}, A[2][0][15] = u_{37}, A[2][1][15] = u_{38}, A[2][2][15] = u_{37} + u_{38},$ $A[2][0][16] = u_{39}, A[2][1][16] = u_{40}, A[2][2][16] = u_{39} + u_{40}, A[2][0][20] = u_{41},$ $A[2][1][20] = u_{41}, A[2][1][21] = u_{42}, A[2][2][21] = u_{42}, A[2][0][24] = u_{43},$ $A[2][1][24] = u_{44}, A[2][2][24] = u_{43} + u_{44}, A[2][0][25] = u_{45}, A[2][1][25] = u_{46},$ $A[2][2][25] = u_{45} + u_{46}, A[2][1][26] = u_{47}, A[2][2][26] = u_{47}, A[2][0][28] = u_{48},$ $A[2][1][28] = u_{49}, A[2][2][28] = u_{48} + u_{49}, A[2][0][29] = u_{50}, A[2][1][29] = u_9 + u_{50},$ $A[2][2][29] = u_9, A[2][0][30] = u_{51}, A[2][1][30] = u_{52}, A[2][2][30] = u_{51} + u_{52},$ $A[3][0][0] = u_{53}, A[3][1][0] = u_{53}, A[3][0][3] = u_{54}, A[3][1][3] = u_{54},$ $A[3][0][4] = u_{55}, A[3][1][4] = u_{55}, A[3][0][5] = u_{56}, A[3][1][5] = u_{56},$ $A[3][0][8] = u_{57}, A[3][1][8] = u_{57}, A[3][0][9] = u_{58}, A[3][1][9] = u_{58},$ $A[3][0][13] = u_{10} + u_{17}, A[3][1][13] = u_{10} + u_{17}, A[3][0][17] = u_{59}, A[3][1][17] = u_{59},$ $A[3][0][19] = u_{60}, A[3][1][19] = u_{60}, A[3][0][22] = u_{61}, A[3][1][22] = u_{61},$ $A[3][0][23] = u_{62}, A[3][1][23] = u_{62}, A[3][0][31] = u_5 + u_{13} + u_{47} + u_{59},$ $A[3][1][31] = u_5 + u_{13} + u_{47} + u_{59},$
Guessed Key Bits
$k_4 + k_{13} + k_{50} + k_{132} + k_{141} + k_{260} + k_{269}, k_{13} + k_{22} + k_{59} + k_{141} + k_{150} + k_{269} + k_{278}$ $k_{18} + k_{27} + k_{146} + k_{155} + k_{160} + k_{274} + k_{283}, k_2 + k_{25} + k_{130} + k_{153} + k_{258} + k_{281} + k_{295}$ $k_2 + k_{11} + k_{130} + k_{139} + k_{258} + k_{267} + k_{304},$

To verify the correctness of our results for 6-round Xoodyak, by parallelism, we have run 76 experiments, which the key was generated randomly, and the correct value of the bit condition was imported at the initial state $A[0][2][21]$. For all of the experiments the zero cube summation was obtained.

5.2 7 Rounds Conditional Cube Attack on Xoodyak

The round function of the Xoodoo is reduced to 7 rounds. By linearizing 1 round of Xoodoo, the degree of the 7-th round is 64. Therefore, we need 65 cube variables to mount the attack, 2 of them build the kernel quadratic term and the rest are ordinary cube variables. Table 3 presents kernel quadratic term, ordinary cube variables, and bit conditions obtained by the MILP model. The process of key recovery is analogous to Algorithm 1.

According to the Table 3, the key-bits need to be recovered are as $k_4 + k_{13} + k_{50} + k_{132} + k_{141} + k_{260} + k_{269}, k_{13} + k_{22} + k_{59} + k_{141} + k_{150} + k_{269} + k_{278}, k_{18} + k_{27} + k_{146} + k_{155} + k_{160} + k_{274} + k_{283}, k_2 + k_{25} + k_{130} + k_{153} + k_{258} + k_{281} + k_{295}, k_2 + k_{11} + k_{130} + k_{139} + k_{258} + k_{267} + k_{304}$. The time complexity to recover a 5-bit key is $2^5 \times 2^{65} = 2^{70}$. By rotating all of the parameters in 64 times, 320 key bits with the complexity of $2^6 \times 2^{70} = 2^{76}$ can be recovered. If we leave the remaining 64 key bits to exhaustive search, the total complexity is $2^{76} + 2^{64} = 2^{76.003}$. This is the first proposed key recovery on 7 rounds of Xoodyak.

6 Conclusion

In this paper, we present a 6-round and 7-round key recovery attack on Xoodyak which is one of the finalists of the lightweight cryptographic algorithm NIST competition with the structure, similar to SHA3 algorithm. As no attack is reported on the full version of Xoodyak, so far, Xoodyak can be regarded as a secure and efficient lightweight algorithm that is still of interest. We use MILP-based tool to find cube variables required to perform the attack and we import our cube variables in the absorbing associated data phase. The complexity of the 6-round attack and 7-round are $2^{42.45}$ and $2^{76.003}$, respectively.

Acknowledgments. The authors would like to thank Prof. Joan Daemen for drawing our attention to the associated data absorption phase of Xoodyak as a suitable target for analysis and for several fruitful discussions. We also thank Dr. Ling Song for explaining and clarifying some of the technical parts of her previous work.

References

1. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Keyak. https://keccak.team/keyak.html
2. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V., Keer, R.V.: The Ketje authenticated encryption scheme. https://keccak.team/ketje.html

3. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Duplexing the sponge: single-pass authenticated encryption and other applications. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 320–337. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28496-0_19
4. Bi, W., Dong, X., Li, Z., Zong, R., Wang, X.: MILP-aided cube-attack-like cryptanalysis on Keccak keyed modes. Des. Codes Cryptogr. **87**(6), 1271–1296 (2019). https://doi.org/10.1007/s10623-018-0526-x
5. Daemen, J., Hoffert, S., Assche, G.V., Keer, R.V.: The design of Xoodoo and Xoofff. IACR Trans. Symmetric Cryptol. **2018**(4), 1–38 (2018). https://doi.org/10.13154/tosc.v2018.i4.1-38
6. Daemen, J., Hoffert, S., Peeters, M., Van Assche, G., Van Keer, R.: Xoodyak, a lightweight cryptographic scheme. IACR Trans. Symmetric Cryptol. **2020**(S1), 60–87 (2020). https://doi.org/10.13154/tosc.v2020.iS1.60-87, https://tosc.iacr.org/index.php/ToSC/article/view/8618
7. Dinur, I., Morawiecki, P., Pieprzyk, J., Srebrny, M., Straus, M.: Cube attacks and cube-attack-like cryptanalysis on the round-reduced Keccak sponge function. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 733–761. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_28
8. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_16
9. Dinur, I., Shamir, A.: Breaking grain-128 with dynamic cube attacks. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 167–187. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21702-9_10
10. Division, N.C.S.: SHA-3 standard: permutation-based hash and extendable-output functions. FIPS Publication 202, National Institute of Standards and Technology, U.S. Department of Commerce (2014). http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf
11. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2: lightweight authenticated encryption and hashing. J. Cryptol. **34**(3), 33 (2021). https://doi.org/10.1007/s00145-021-09398-9
12. Dong, X., Li, Z., Wang, X., Qin, L.: Cube-like attack on round-reduced initialization of Ketje Sr. IACR Trans. Symmetric Cryptol. **2017**(1), 259–280 (2017). https://doi.org/10.13154/tosc.v2017.i1.259-280
13. Huang, S., Wang, X., Xu, G., Wang, M., Zhao, J.: Conditional cube attack on reduced-round Keccak sponge function. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part II. LNCS, vol. 10211, pp. 259–288. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_9
14. Lai, X.: Higher order derivatives and differential cryptanalysis. In: Blahut, R.E., Costello, D.J., Maurer, U., Mittelholzer, T. (eds.) Communications and Cryptography. The Springer International Series in Engineering and Computer Science, vol. 276, pp. 227–233. Springer, Boston (1994). https://doi.org/10.1007/978-1-4615-2694-0_23
15. Li, Z., Bi, W., Dong, X., Wang, X.: Improved conditional cube attacks on Keccak keyed modes with MILP method. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 99–127. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_4
16. Li, Z., Dong, X., Bi, W., Jia, K., Wang, X., Meier, W.: New conditional cube attack on Keccak keyed modes. IACR Cryptol. ePrint Arch. 392 (2019). https://eprint.iacr.org/2019/392

17. National Institute of Standards and Technology (NIST): Lightweight Cryptography Standardization Process: NIST Selects Ascon. NIST Website (2023). https://csrc.nist.gov/News/2023/lightweight-cryptography-nist-selects-ascon
18. National Institute of Standards and Technology (NIST): Lightweight Cryptography. NIST Website (2016). https://www.nist.gov/programs-projects/lightweight-cryptography
19. Song, L., Guo, J.: Cube-attack-like cryptanalysis of round-reduced Keccak using MILP. IACR Trans. Symmetric Cryptol. **2018**(3), 182–214 (2018). https://doi.org/10.13154/tosc.v2018.i3.182-214
20. Song, L., Guo, J., Shi, D., Ling, S.: New MILP modeling: improved conditional cube attacks on Keccak-based constructions. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11273, pp. 65–95. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03329-3_3
21. The U.S. National Institute of Standards and Technology (NIST): SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash (2016). http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf
22. Zhou, H., Li, Z., Dong, X., Jia, K., Meier, W.: Practical key-recovery attacks on round-reduced Ketje Jr. Xoodoo-AE and Xoodyak. Comput. J. **63**(8), 1231–1246 (2020). https://doi.org/10.1093/comjnl/bxz152